Introduction to Program Analysis Techniques with Applications to Smart Contract Security

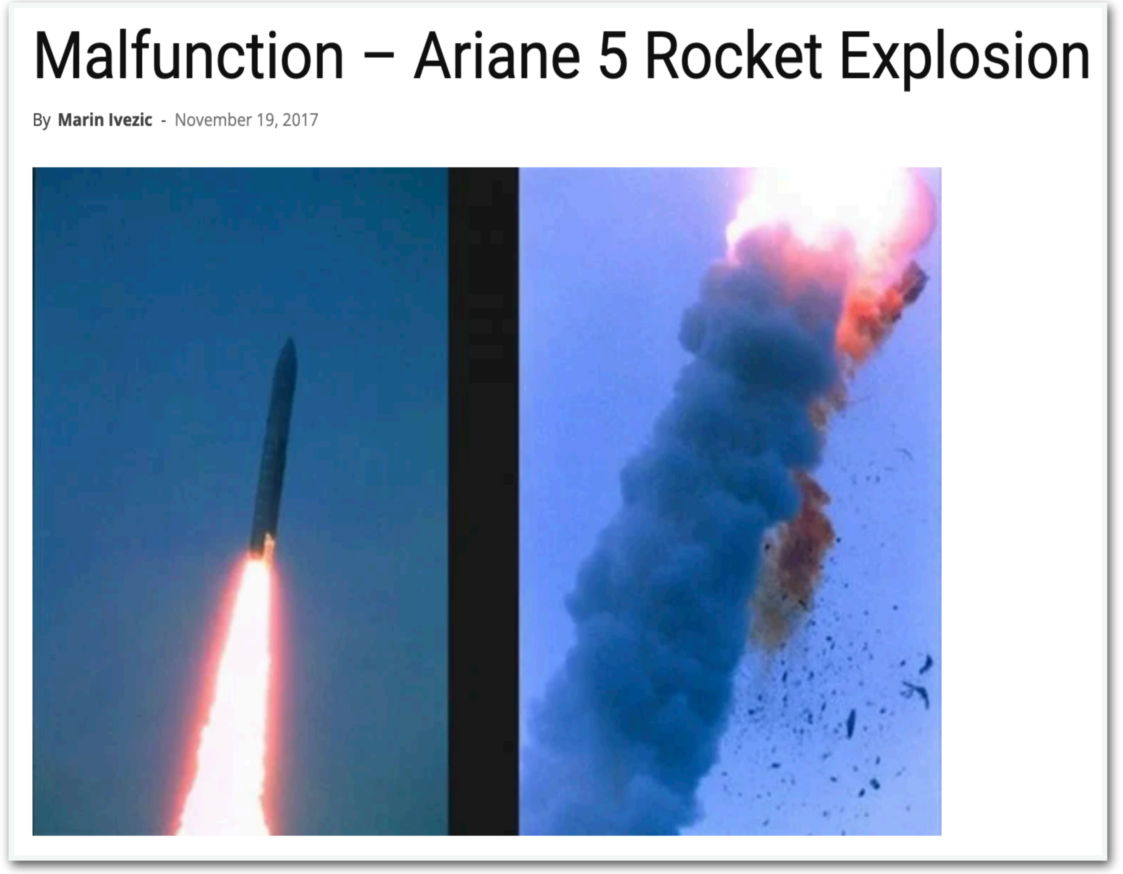
Sunbeom So (소순범)
Assistant Professor, GIST

8 September 2023 @ GIST EECS Colloquium

Research Area: PL / SW Security / SE

- Solving problems in SW Security and SW Engineering using program analysis.
- **Published 1st author papers at Top conferences**: 5 papers (BK IF 3,4)
  - **FSE 2023 (BK IF 4)**: fixing vulnerabilities — Software Engineering Top
  - **ICSE 2023 (BK IF 4)**: finding correctness bugs — Software Engineering Top
  - **USENIX Security 2021 (BK IF 3)**: finding vulnerabilities — Security Top
  - **IEEE S&P 2020 (BK IF 4)**: vulnerability-safety verification — Security Top
  - **IJCAI 2018 (BK IF 4)**: program synthesis — AI Top

Motivation: SW Bugs are Everywhere


Malfunction – Ariane 5 Rocket Explosion

By Marin Ivezic · November 19, 2017

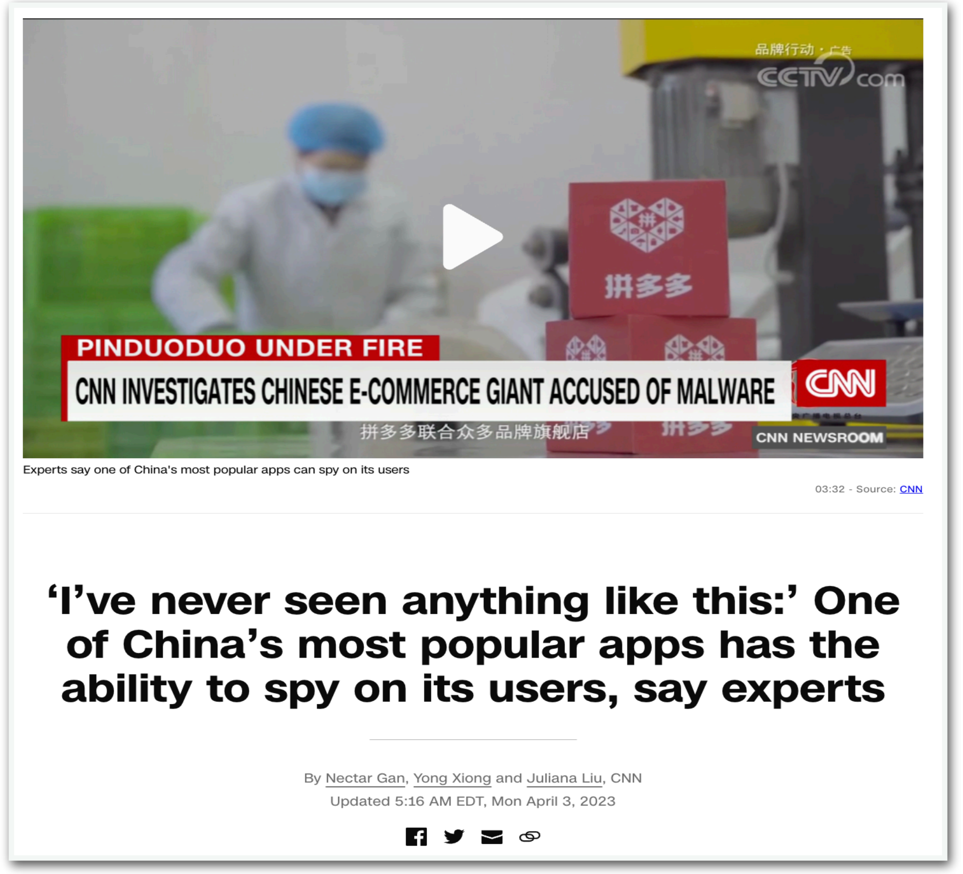
Rocket SW


Airline Blames Bad Software in San Francisco Crash

Give this article

The wreckage of Asiana Flight 214 last year. Boeing has focused on the crew's failure to maintain proper airspeed in the crash, but in a filing, Asiana also blamed "inconsistencies with the aircraft's automation logic." Marcio Jose Sanchez/Associated Press

Aircraft SW



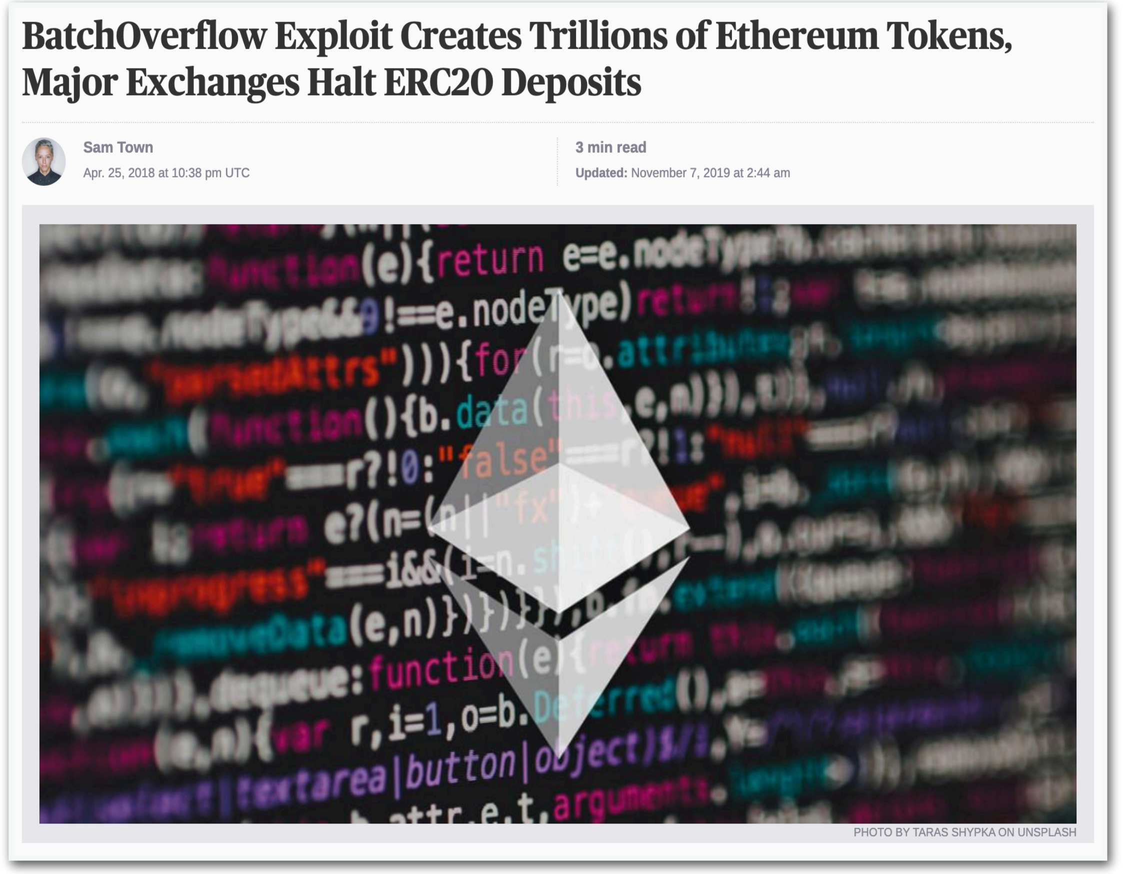
Experts say one of China's most popular apps can spy on its users

'I've never seen anything like this:' One of China's most popular apps has the ability to spy on its users, say experts

By Nectar Gan, Yong Xiong and Juliana Liu, CNN

Updated 5:16 AM EDT, Mon April 3, 2023

Mobile App


BatchOverflow Exploit Creates Trillions of Ethereum Tokens, Major Exchanges Halt ERC20 Deposits

Sam Town

3 min read

Blockchain SW


Zeljka Zorz, Editor-in-Chief, Help Net Security

January 30, 2023

Share

Critical OpenEMR vulnerabilities may allow attackers to access patients' health records

Critical vulnerabilities discovered in OpenEMR can be chained to gain code execution on a server running a vulnerable version of the popular open-source electronic health record system.

Medical SW


Tesla recalls 362,000 vehicles over self-driving software flaws that risk crashes

Regulators say driver assistance system does not adequately adhere to traffic safety laws and can cause crashes

Tesla Model 3 compact full electric car with a full self-driving system. Photograph: Sjoerd van der Wal/Getty Images

Self-driving SW

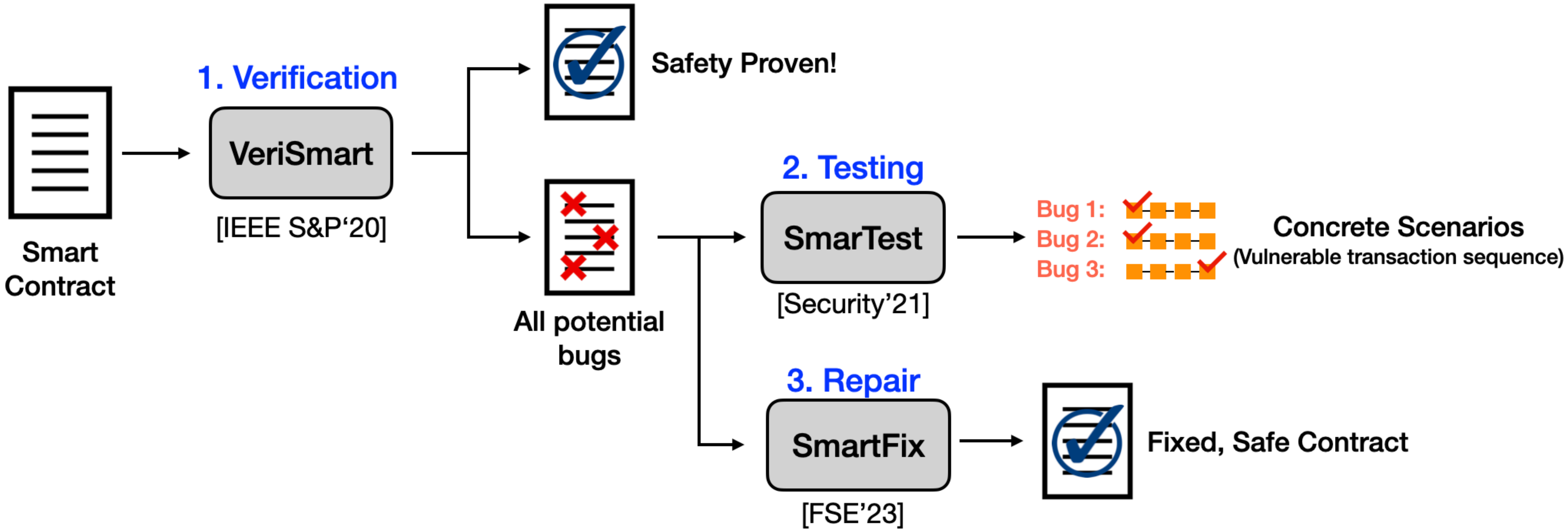
Goal: Prevent Disasters due to SW Bugs

- **Program Verification** for automatically proving the absence of bugs
  - VeriSmart [IEEE S&P’20]
- **Program Testing** for automatically finding bugs
  - SmarTest [USENIX Security’21], Diver [ICSE’23]
- **Program Repair** for automatically fixing bugs
  - FixML [OOPSLA’18], SmartFix [FSE’23]
- **Program Synthesis** for automatically generating programs
  - AlphaRegex [GPCE’16], SIMPL [SAS’17], PAT [IJCAI’18]

Demo

This Talk:
Program Analysis for Smart Contract Security

1. **VeriSmart** (IEEE S&P 2020): program verification for smart contracts
2. **SmarTest** (USENIX Security 2021): program testing for smart contracts
3. **SmartFix** (FSE 2023): program repair for smart contracts



1. VeriSmart: Program Verification

VeriSmart: A Highly Precise Safety Verifier for Ethereum Smart Contracts

IEEE Symposium on Security and Privacy 2020

Smart Contract

- Digital contract written in programming languages.

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

Solidity Contract

Smart Contract

- Digital contract written in programming languages.

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

State (global) variables

Constructor

Function

Function

Solidity Contract

Smart Contract

- Transaction execution = Function invocation

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

Solidity Contract

X sends 5 tokens to Y.

balance[X] = 20,
balance[Y] = 0

transfer(Y, 5)
with X=msg.sender

balance[X] = 15,
balance[Y] = 5

Importance of Safe Smart Contracts

- **Immutable** once deployed on blockchains.
- **Huge financial damage** once exploited.

(2016)

KLINT FINLEY 06.18.16 04:30 AM

A $50 Million Hack Just Showed That the DAO Was All Too Human

(2018)

ETHEREUM › TECHNOLOGY

BatchOverflow Exploit Creates Trillions of Ethereum Tokens, Major Exchanges Halt ERC20 Deposits

Sam Town · April 25, 2018 at 10:38 pm UTC · 3 min read

(2021)

DIGITAL HEIST —

Really stupid “smart contract” bug let hackers steal $31 million in digital coin

Company says it has contacted the hacker in an attempt to recover the funds. Good luck.

DAN GOODIN - 12/2/2021, 8:41 AM

(2023)

SushiSwap Smart Contract Bug Exploited in $3.3 Million Theft

The decentralized exchange says it's "all hands on deck" and that some of the funds have been recovered.

By Ryan Ozawa

Apr 10, 2023
2 min read

Goal: Ensuring safety before deployment

SmartMesh (CVE-2018-10376)

CVE-ID	
CVE-2018-10376	[Learn more at National Vulnerability Database (NVD)](#) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
An integer overflow in the transferProxy function of a smart contract implementation for SmartMesh (aka SMT), an Ethereum ERC20 token, allows attackers to accomplish an unauthorized increase of digital assets via crafted _fee and _value parameters, as exploited in the wild in April 2018, aka the "proxyOverflow" issue.	

ERC-20 Tokens Transferred: 2

From 0xDF31A4...B34Eb46F **To** 0xDF31A4...B34Eb46F **For**
65,133,050,195,990,359,925,758,679,067,386,948,167,464,366,374,422,817,272,194.891004451135422463 **6.29651431288701E+57**
SmartMesh... (SMT...)

From 0xDF31A4...B34Eb46F **To** 0xd6a09B...e2e0651E **For**
50,659,039,041,325,835,497,812,305,941,300,959,685,805,618,291,217,746,767,262.693003461994217473 **4.89728891002323E+57**
SmartMesh... (SMT...)

https://etherscan.io/tx/0x1abab4c8db9a30e703114528e31dee129a3a758f7f8abc3b6494aad3d304e43f

SmartMesh (CVE-2018-10376)

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

SmartMesh (CVE-2018-10376)

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

SmartMesh (CVE-2018-10376)

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
       balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

Send money to the receiver (to)

SmartMesh (CVE-2018-10376)

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

Send money to the receiver (`to`)

Pay fee to the proxy (`msg.sender`)

SmartMesh (CVE-2018-10376)

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

Send money to the receiver (`to`)

Pay fee to the proxy (`msg.sender`)

Deduct money from the sender (`from`)

SmartMesh (CVE-2018-10376)

```
1    function transferProxy (address from, address to, uint value, uint fee)
2    public returns (bool) {
3         if (balance[from] < fee + value)
4           revert ();
5
6         if (balance[to] + value < balance[to] ||
7             balance[msg.sender] + fee < balance[msg.sender])
8           revert ();
9
10        balance[to] += value;
11        balance[msg.sender] += fee;
12        balance[from] -= value + fee;
13        return true;
14   }
```

To prevent underflows in the token sender's balance (L12)

Send money to the receiver (`to`)

Pay fee to the proxy (`msg.sender`)

Deduct money from the sender (`from`)

SmartMesh (CVE-2018-10376)

```
1    function transferProxy (address from, address to, uint value, uint fee)
2    public returns (bool) {
3        if (balance[from] < fee + value)
4          revert ();
5
6        if (balance[to] + value < balance[to] ||
7            balance[msg.sender] + fee < balance[msg.sender])
8          revert ();
9
10       balance[to] += value;
11       balance[msg.sender] += fee;
12       balance[from] -= value + fee;
13       return true;
14   }
```

To prevent underflows in the token sender's balance (L12)

To prevent overflows in the token recipient's balance (L10, L11)

Send money to the receiver (`to`)

Pay fee to the proxy (`msg.sender`)

Deduct money from the sender (`from`)

SmartMesh (CVE-2018-10376)

Q. Is this function well-written?

```
1   function transferProxy (address from, address to, uint value, uint fee)
2   public returns (bool) {
3       if (balance[from] < fee + value)
4         revert ();
5
6       if (balance[to] + value < balance[to] ||
7           balance[msg.sender] + fee < balance[msg.sender])
8         revert ();
9
10      balance[to] += value;
11      balance[msg.sender] += fee;
12      balance[from] -= value + fee;
13      return true;
14  }
```

To prevent underflows in the token sender’s balance (L12)

To prevent overflows in the token recipient’s balance (L10, L11)

Send money to the receiver (`to`)

Pay fee to the proxy (`msg.sender`)

Deduct money from the sender (`from`)

SmartMesh (CVE-2018-10376)

Q. Is this function well-written?

may overflow to 0!

```
1   function transferProxy (address from, address to, uint value, uint fee)
2   public returns (bool) {
3       if (balance[from] < fee + value)
4         revert ();
5
6       if (balance[to] + value < balance[to] ||
7           balance[msg.sender] + fee < balance[msg.sender])
8         revert ();
9
10      balance[to] += value;
11      balance[msg.sender] += fee;
12      balance[from] -= value + fee;
13      return true;
14  }
```

To prevent underflows in the token sender's balance (L12)

To prevent overflows in the token recipient's balance (L10, L11)

Send money to the receiver (`to`)

Pay fee to the proxy (`msg.sender`)

Deduct money from the sender (`from`)

SmartMesh (CVE-2018-10376)

256-bit unsigned integers in hexadecimal numbers (64 digits)

```
balance[from] = balance[to] = balance[msg.sender] = 0

value = 0x8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee   = 0x7000000000000000000000000000000000000000000000000000000000000001
```

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

SmartMesh (CVE-2018-10376)

256-bit unsigned integers in hexadecimal numbers (64 digits)

```
balance[from] = balance[to] = balance[msg.sender] = 0

value = 0x8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee   = 0x7000000000000000000000000000000000000000000000000000000000000001
```

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)   false!
      revert ();                0

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

SmartMesh (CVE-2018-10376)

256-bit unsigned integers in hexadecimal numbers (64 digits)

```
balance[from] = balance[to] = balance[msg.sender] = 0

value = 0x8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee   = 0x7000000000000000000000000000000000000000000000000000000000000001
```

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

false!

0

0

false!

0

SmartMesh (CVE-2018-10376)

256-bit unsigned integers in hexadecimal numbers (64 digits)

```
balance[from] = balance[to] = balance[msg.sender] = 0

value = 0x8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee   = 0x7000000000000000000000000000000000000000000000000000000000000001
```

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)          false!
                         0
      revert ();
                                   0
    if (balance[to] + value < balance[to] ||
     balance[msg.sender] + fee < balance[msg.sender])   false!
                                          0
      revert ();

    balance[to] += value;          0x8fff…ffff
    balance[msg.sender] += fee;    0x7000…0001
    balance[from] -= value + fee;  0
    return true;
}
```

Limitations of Existing Safety Analyzers

- **Bug-finders** could fail to find critical bugs.
  - E.g., Osiris [ACSAC ’18], Oyente [CCS ’16], Mythril, Manticore

```
function transferProxy (address from, address to, uint value, uint fee)
public returns (bool) {
    if (balance[from] < fee + value)
      revert ();

    if (balance[to] + value < balance[to] ||
        balance[msg.sender] + fee < balance[msg.sender])
      revert ();

    balance[to] += value;
    balance[msg.sender] += fee;
    balance[from] -= value + fee;
    return true;
}
```

Only Osiris detects this vulnerability

CVE-2018-10376

Limitations of Existing Safety Analyzers

- **Bug-finders** could fail to find critical bugs.
  - E.g., Osiris [ACSAC ’18], Oyente [CCS ’16], Mythril, Manticore

```
function multipleTransfer (address[] to, uint value) public returns (bool) {
    require(value * to.length > 0);
    require(balance[msg.sender] >= value * to.length);
    balance[msg.sender] -= value * to.length;

    for (uint i =0; i < to.length; ++i) {
      balance[to[i]] += value;
    }
    return true;
}
```

Despite the similarity, Osiris now fails!

CVE-2018-14006

Limitations of Existing Safety Analyzers

- **Verifiers** typically suffer from lots of false positives.
  - E.g., Zeus [NDSS ’18], SMTChecker [ISoLA ’18]

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

False alarm by Zeus, SMTChecker

False alarm by Zeus, SMTChecker

Limitations of Existing Safety Analyzers

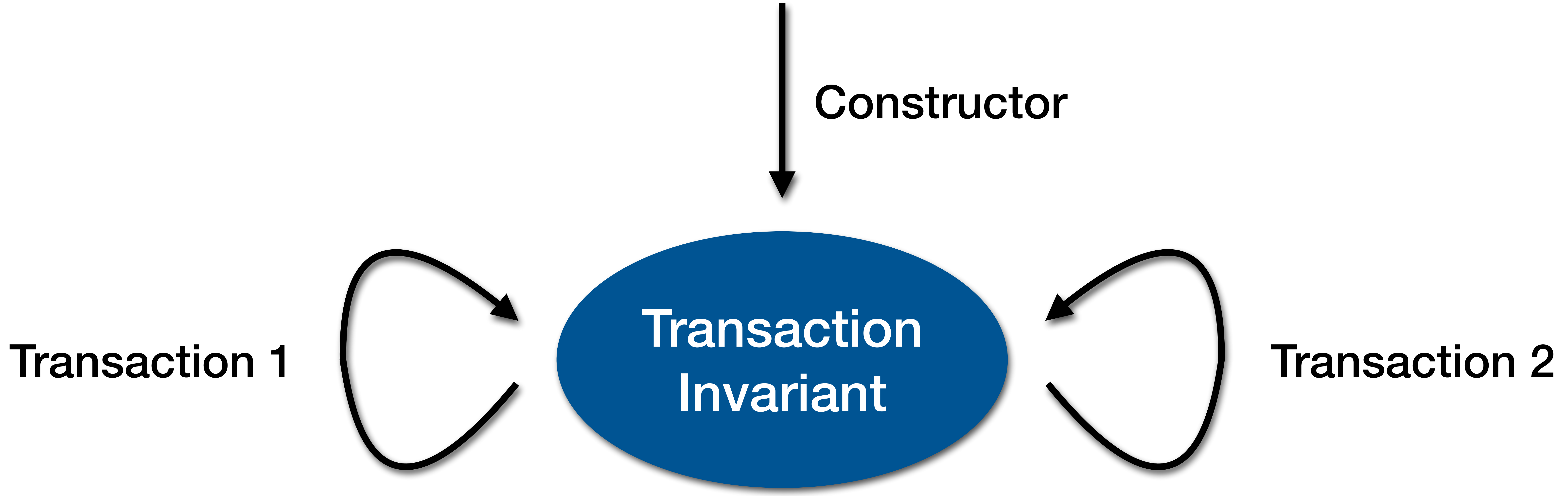
- **Bug-finders** could fail to find critical vulnerabilities.
  - Consider a subset of program paths.
  - E.g., Osiris [ACSAC ’18], Oyente [CCS ’16], Mythril, Manticore
- **Verifiers** suffer from lots of false positives.
  - Imprecise reasoning on global properties of smart contracts.
  - E.g., Zeus [NDSS ’18], SMTChecker [ISoLA ’18]

VeriSmart’s Goal: overcome shortcomings of existing analyzers.

VeriSmart's Basic Technology: Program Verification

- Express a program and a property in first-order logic formulas (Φ_P, Φ_{safe}).
- Determine the absence of bugs by checking the satisfiability of $\Phi_P \land \neg\Phi_{safe}$.

VeriSmart's Key Feature: Automatic Inference and Use of Transaction Invariant



- Global property that holds under arbitrary interleaving of transactions.
  1. Valid at the exit of the constructor.
  2. Validity preserved by executions of transactions.

VeriSmart's Key Feature:
Automatic Inference and Use of Transaction Invariant

$totalSupply = \Sigma_i balance[i]$

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

False alarm by Zeus, SMTChecker

False alarm by Zeus, SMTChecker

VeriSmart's Key Feature: Automatic Inference and Use of Transaction Invariant

$totalSupply = \Sigma_i balance[i]$

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

$totalSupply = \Sigma_i balance[i]$

False alarm by Zeus, SMTChecker

False alarm by Zeus, SMTChecker

VeriSmart's Key Feature:
Automatic Inference and Use of Transaction Invariant

$totalSupply = \Sigma_i balance[i]$

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

$totalSupply = \Sigma_i balance[i]$

$totalSupply = \Sigma_i balance[i]$

$totalSupply = \Sigma_i balance[i]$

False alarm by Zeus, SMTChecker

False alarm by Zeus, SMTChecker

VeriSmart's Key Feature: Automatic Inference and Use of Transaction Invariant

totalSupply = $\Sigma_i$balance[i]

```
contract Netkoin {
  mapping (address => uint) public balance;
  uint public totalSupply;

  constructor (uint initialSupply) {
    totalSupply = initialSupply;
    balance[msg.sender] = totalSupply;
  }

  function transfer (address to, uint value) public
  returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    balance[to] += value;
    return true;
  }

  function burn(uint value) public returns (bool) {
    require (balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    totalSupply -= value;
    return true;
  }
}
```

False alarm by Zeus, SMTChecker

False alarm by Zeus, SMTChecker

totalSupply = $\Sigma_i$balance[i]

totalSupply = $\Sigma_i$balance[i]

totalSupply = $\Sigma_i$balance[i]

totalSupply = $\Sigma_i$balance[i]

totalSupply = $\Sigma_i$balance[i]

Precise Verification with Transaction Invariants

totalSupply = $\Sigma_i$balance[i]

```
18    function burn(uint value) public returns (bool) {
19      require (balance[msg.sender] >= value);
20      balance[msg.sender] -= value;
21      totalSupply -= value;
22      return true;
23    }
24  }
```

totalSupply = $\Sigma_i$balance[i]

False alarm by Zeus, SMTChecker

- To show: totalSupply >= value holds at line 21.

Precise Verification with Transaction Invariants

$totalSupply = \Sigma_i balance[i]$

```
18    function burn(uint value) public returns (bool) {
19      require (balance[msg.sender] >= value);
20      balance[msg.sender] -= value;
21      totalSupply -= value;
22      return true;
23    }
24  }
```

$totalSupply = \Sigma_i balance[i]$

False alarm by Zeus, SMTChecker

- To show: totalSupply >= value holds at line 21.
- Verification using the inferred invariant:

totalSupply >= balance[msg.sender]

from "$totalSupply = \Sigma_i balance[i]$"

Precise Verification with Transaction Invariants

totalSupply = $\Sigma_i$balance[i]

totalSupply = $\Sigma_i$balance[i]

```
18      function burn(uint value) public returns (bool) {
19        require (balance[msg.sender] >= value);
20        balance[msg.sender] -= value;
21        totalSupply -= value;
22        return true;
23      }
24    }
```

False alarm by Zeus, SMTChecker

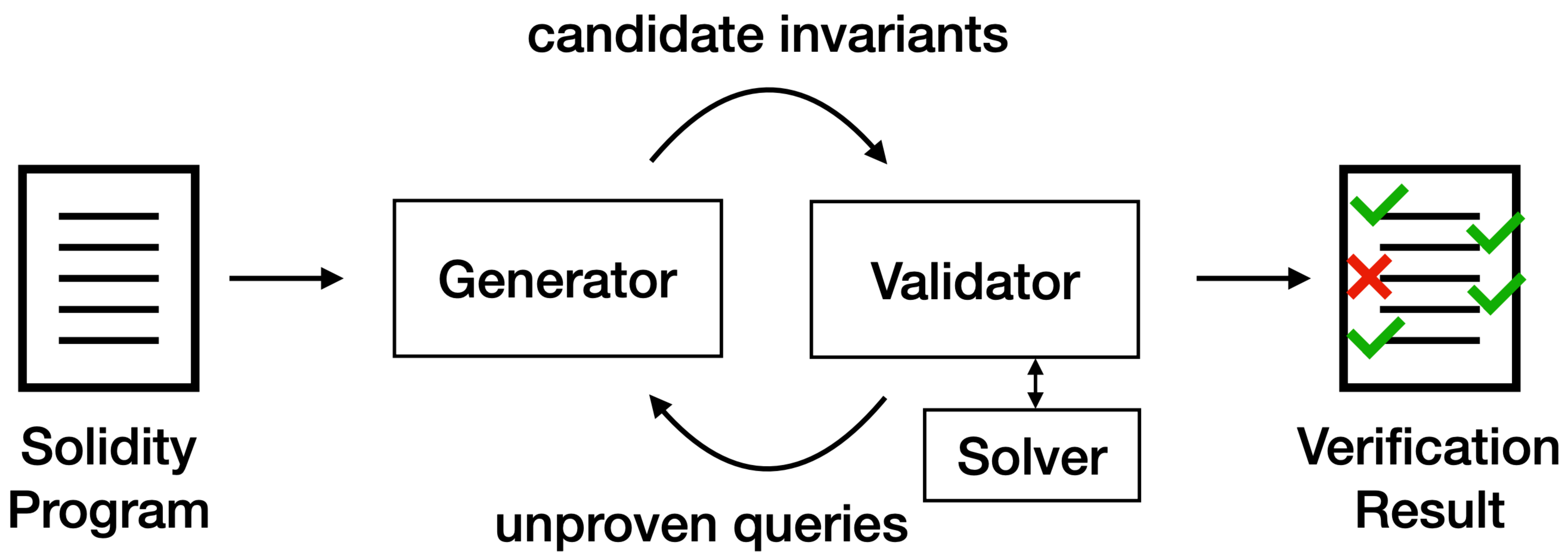
- To show: totalSupply >= value holds at line 21.
- Verification using the inferred invariant:

totalSupply >= balance[msg.sender] >= value

from “totalSupply = $\Sigma_i$balance[i]”

line 19

VeriSmart Algorithm



- **Generator:** produce transaction/loop candidate invariants.
- **Validator:** validate the contract using candidate invariants.

Evaluation: vs. Bug-finders

	VeriSmart	Osiris [ACSAC '18]	Oyente [CCS '16]	Mythril (Consensys)	Manticore (Trail of Bits)
#Alarm	492	240	171	94	14
#False Positive	2	13	14	10	0
Recall (#valid CVE: 58)	**100%** >>	70.7%	60.3%	19.0%	3.4%
Precision (#TP/ #Alarm)	**99.6%** >>	94.6%	91.8%	89.4%	n/a (too low recall)

Results on 60 contracts with CVE vulnerabilities

Evaluation: vs. Bug-finders

	VeriSmart	Osiris [ACSAC '18]	Oyente [CCS '16]	Mythril (Consensys)	Manticore (Trail of Bits)
#Alarm	492	240	171	94	14
#False Positive	2	13	14	10	0
Recall (#valid CVE: 58)	**100%** >>	70.7%	60.3%	19.0%	3.4%
Precision (#TP/ #Alarm)	**99.6%** >>	94.6%	91.8%	89.4%	n/a (too low recall)

Results on 60 contracts with CVE vulnerabilities

Evaluation: vs. Bug-finders

	VeriSmart	Osiris [ACSAC '18]	Oyente [CCS '16]	Mythril (Consensys)	Manticore (Trail of Bits)
#Alarm	492	240	171	94	14
#False Positive	2	13	14	10	0
Recall (#valid CVE: 58)	100% >>	70.7%	60.3%	19.0%	3.4%
Precision (#TP/ #Alarm)	99.6% >>	94.6%	91.8%	89.4%	n/a (too low recall)

Results on 60 contracts with CVE vulnerabilities

Evaluation: vs. Verifiers

- **Benchmark:** 25 contracts where Zeus[NDSS'18] produced false positives.
  - VeriSmart verified 24 contracts without false positives!

No.	LOC	#Q	VeriSmart #Alarm	VeriSmart #FP	VeriSmart Verified	SMTChecker [12] #Alarm	SMTChecker [12] #FP	SMTChecker [12] Verified	Zeus [11] Verified
#1	42	3	0	0	✓	3	3	✗	✗
#2	78	2	1	0	✓	2	1	✗	✗
#3	75	7	2	0	✓	7	5	✗	✗
#4	70	7	0	0	✓	7	7	✗	✗
#5	103	8	0	0	✓	6	6	✗	✗
#6	141	5	2	0	✓	internal error			✗
#7	74	6	1	0	✓	6	5	✗	✗
#8	84	6	0	0	✓	4	4	✗	✗
#9	82	6	0	0	✓	6	6	✗	✗
#10	99	2	1	0	✓	internal error			✗
#11	171	15	9	0	✓	internal error			✗
#12	139	7	0	0	✓	internal error			✗
#13	139	7	0	0	✓	internal error			✗
#14	139	7	0	0	✓	internal error			✗
#15	139	7	0	0	✓	internal error			✗
#16	141	16	10	0	✓	internal error			✗
#17	153	5	0	0	✓	internal error			✗
#18	139	7	0	0	✓	internal error			✗
#19	113	4	0	0	✓	4	4	✗	✗
#20	40	3	0	0	✓	3	3	✗	✗
#21	59	3	0	0	✓	internal error			✗
#22	28	3	1	0	✓	1	0	✓	✗
#23	19	3	0	0	✓	3	3	✗	✗
#24	457	30	13	6	✗	internal error			✗
#25	17	3	0	0	✓	3	3	✗	✗
Total	2741	172	40	6	✓:24 ✗ : 1	55	50	✓: 1 ✗: 12	✓: 0 ✗:25

✓ : No false positives

✗ : False positives exist

Evaluation: Importance of Transaction Invariants

only 8 contracts

- Without transaction invariants, VeriSmart verified ~~24 contracts~~ without false positives.
  - On 25 contracts from the Zeus [NDSS'18] dataset.
- VeriSmart found 6 incorrectly-reported CVE vulnerabilities.
  - By proving the safety at locations known to be vulnerable.

CVE ID	Name	#Incorrect Queries	#FP		
			OSIRIS	OYENTE	VERISMART
2018-13113	ETT	2	2	2	0
2018-13144	PDX	1	1	1	0
2018-13326	BTX	2	2	2	0
2018-13327	CCLAG	1	1	1	0

2. SmarTest: Program Testing

SmarTest : Effectively Hunting Vulnerable Transaction Sequences in Smart Contracts through Language Model-Guided Symbolic Execution
USENIX Security 2021

SmarTest's Goal:
Find Vulnerabilities with Concrete Scenarios

```
1   contract Example {
2     address owner;
3     mapping (address => uint) balance;
4     mapping (address => mapping (address => uint)) allowed;
5     uint totalSupply;
6
7     constructor () public {
8       owner = msg.sender;
9       totalSupply = 0;
10    }
11
12    function mintToken (address target, uint amount) {
13      require (owner == msg.sender);
14      balance[target] += amount;
15      totalSupply += amount;
16    }
17
18    function approve(address spender, uint value)
19    public returns (bool) {
20      allowed[msg.sender][spender] = value;
21      return true;
22    }
23
24    function burnFrom (address from, uint value)
25    public returns (bool) {
26      require (balance[from] >= value);
27      require (allowed[from][msg.sender] >= value);
28      balance[from] -= value;
29      allowed[from][msg.sender] -= value;
30      totalSupply -= value;
31      return true;
32    }
33  }
```

Underflow
(line 30)

Analyzer →

"There may be an underflow vulnerability at L30."

SmarTest's Goal:
Find Vulnerabilities with Concrete Scenarios

```
1   contract Example {
2     address owner;
3     mapping (address => uint) balance;
4     mapping (address => mapping (address => uint)) allowed;
5     uint totalSupply;
6
7     constructor () public {
8       owner = msg.sender;
9       totalSupply = 0;
10    }
11
12    function mintToken (address target, uint amount) {
13      require (owner == msg.sender);
14      balance[target] += amount;
15      totalSupply += amount;
16    }
17
18    function approve(address spender, uint value)
19    public returns (bool) {
20      allowed[msg.sender][spender] = value;
21      return true;
22    }
23
24    function burnFrom (address from, uint value)
25    public returns (bool) {
26      require (balance[from] >= value);
27      require (allowed[from][msg.sender] >= value);
28      balance[from] -= value;
29      allowed[from][msg.sender] -= value;
30      totalSupply -= value;
31      return true;
32    }
33  }
```

Underflow (line 30)

Analyzer

"There may be an underflow vulnerability at L30."

In what situations?

SmarTest's Goal:
Find Vulnerabilities with Concrete Scenarios

```
1   contract Example {
2     address owner;
3     mapping (address => uint) balance;
4     mapping (address => mapping (address => uint)) allowed;
5     uint totalSupply;
6
7     constructor () public {
8       owner = msg.sender;
9       totalSupply = 0;
10    }
11
12    function mintToken (address target, uint amount) {
13      require (owner == msg.sender);
14      balance[target] += amount;
15      totalSupply += amount;
16    }
17
18    function approve(address spender, uint value)
19    public returns (bool) {
20      allowed[msg.sender][spender] = value;
21      return true;
22    }
23
24    function burnFrom (address from, uint value)
25    public returns (bool) {
26      require (balance[from] >= value);
27      require (allowed[from][msg.sender] >= value);
28      balance[from] -= value;
29      allowed[from][msg.sender] -= value;
30      totalSupply -= value;
31      return true;
32    }
33  }
```

Underflow (line 30)

SmarTest

"There ~~may be~~ exists an underflow vulnerability at L30."

"vulnerable transaction sequence" (length 4)

mintToken (A, 1)
with msg.sender = owner

totalSupply = 1,
balance[A] = 1

mintToken (B, 2^{256} - 1)
with msg.sender = owner

totalSupply = 0,
balance[B] = 2^{256} - 1

approve (C, 1)
with msg.sender = B

allowed[B][C] = 1

burnFrom (B, 1)
with msg.sender = C

Overflow at line 15

Can pass line 26

Can pass line 27

Pass line 26, 27
Underflow at line 30

Testing using logical formulas

Basic Approach

- Symbolic execution in increasing order.

```
contract Example {
  bool flag;
  uint x;

  constructor () public {
    flag = false;
    x = 100;
  }

  function setFlag () public {
    flag = true;
  }

  function setX () public {
    require (flag == true);
    x = 0;
  }

  function decX () public {
    require (flag == true);
    require (x < 100);
    x = x - 1;
  }
}
```

Testing using logical formulas

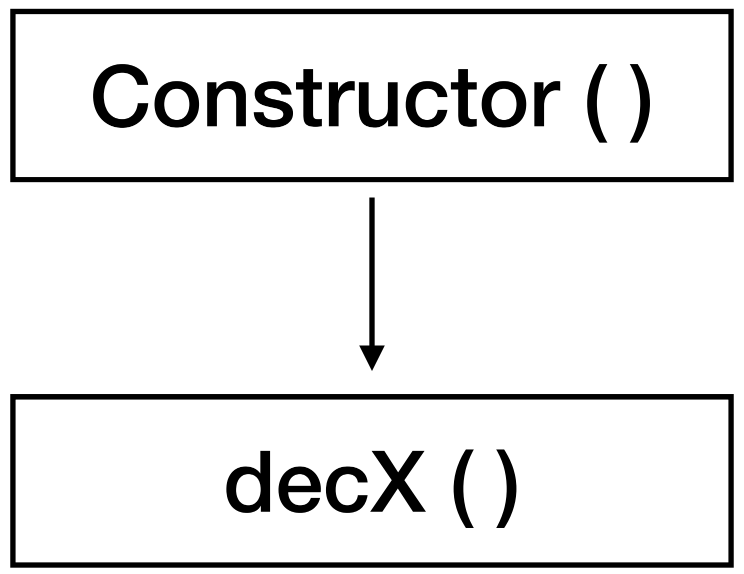
Basic Approach

- Symbolic execution in increasing order.

```
contract Example {
  bool flag;
  uint x;

  constructor () public {
    flag = false;
    x = 100;
  }

  function setFlag () public {
    flag = true;
  }

  function setX () public {
    require (flag == true);
    x = 0;
  }

  function decX () public {
    require (flag == true);
    require (x < 100);
    x = x - 1;
  }
}
```

Goal:
Trigger underflow

Transaction Sequence



Verification Condition

Testing using logical formulas

Basic Approach

- Symbolic execution in increasing order.

```
contract Example {
  bool flag;
  uint x;

  constructor () public {
    flag = false;
    x = 100;
  }

  function setFlag () public {
    flag = true;
  }

  function setX () public {
    require (flag == true);
    x = 0;
  }

  function decX () public {
    require (flag == true);
    require (x < 100);
    x = x - 1;
  }
}
```

Goal:
Trigger underflow

Transaction Sequence

Constructor ( )

↓

decX ( )

Verification Condition

$$(flag = false) \land (x = 100)$$

$$\land\, (flag = true) \land (x < 100) \land \neg(x \geq 1)$$

Negation of
underflow-safety condition

Basic Approach

Testing using logical formulas

- Symbolic execution in increasing order.

```
contract Example {
  bool flag;
  uint x;

  constructor () public {
    flag = false;
    x = 100;
  }

  function setFlag () public {
    flag = true;
  }

  function setX () public {
    require (flag == true);
    x = 0;
  }

  function decX () public {
    require (flag == true);
    require (x < 100);
    x = x - 1;
  }
}
```

Goal:
Trigger underflow

Transaction Sequence

Constructor ( ) → decX ( )

Verification Condition

SMT Solver: UNSAT!

$$(flag = false) \land (x = 100)$$

Conflict!

Conflict!

$$\land\, (flag = true) \land (x < 100) \land \neg(x \geq 1)$$

Negation of underflow-safety condition

Testing using logical formulas

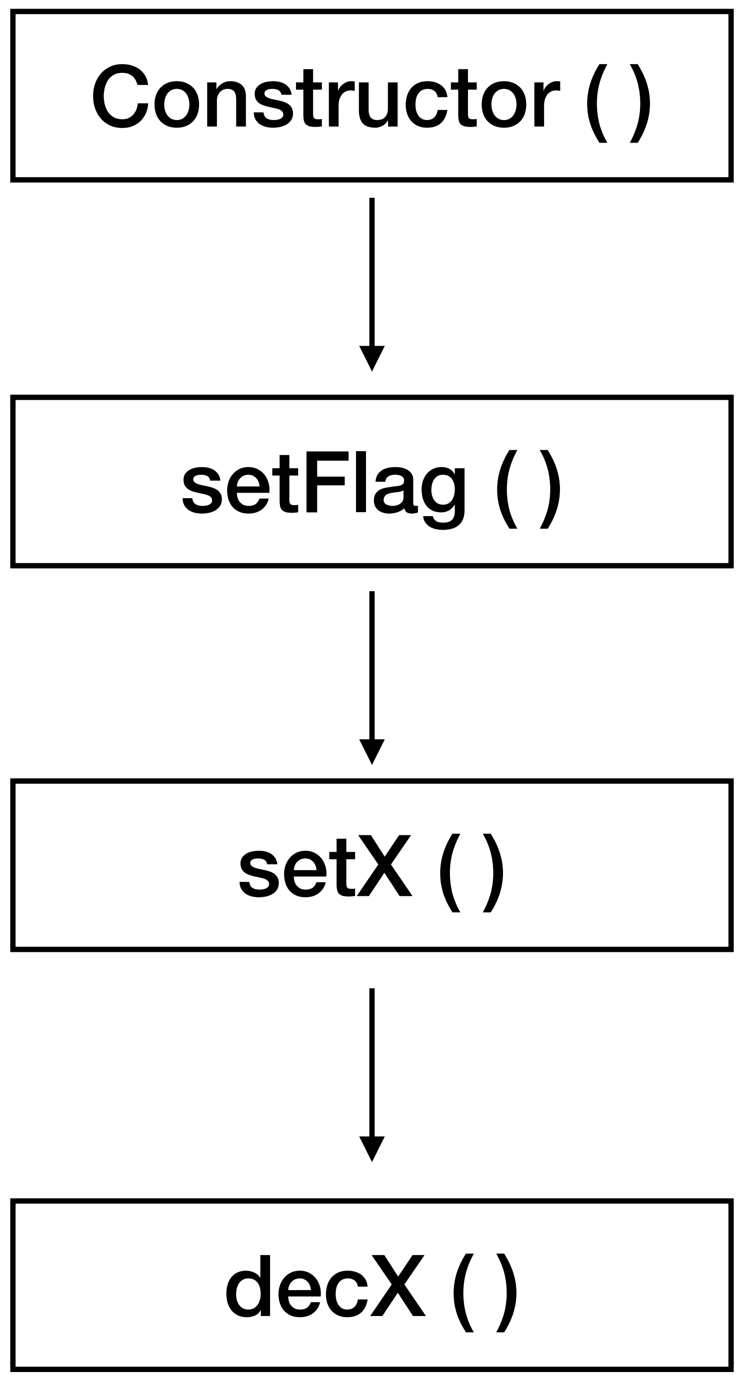
Basic Approach

- Symbolic execution in increasing order.

```
contract Example {
  bool flag;
  uint x;

  constructor () public {
    flag = false;
    x = 100;
  }

  function setFlag () public {
    flag = true;
  }

  function setX () public {
    require (flag == true);
    x = 0;
  }

  function decX () public {
    require (flag == true);
    require (x < 100);
    x = x - 1;
  }
}
```

Goal:
Trigger underflow

Transaction Sequence



Verification Condition

Testing using logical formulas

Basic Approach

- Symbolic execution in increasing order.

```
contract Example {
  bool flag;
  uint x;

  constructor () public {
    flag = false;
    x = 100;
  }

  function setFlag () public {
    flag = true;
  }

  function setX () public {
    require (flag == true);
    x = 0;
  }

  function decX () public {
    require (flag == true);
    require (x < 100);
    x = x - 1;
  }
}
```

Goal:
Trigger underflow

Transaction Sequence	Verification Condition
Constructor ( )	$(flag' = false) \wedge (x' = 100)$
↓ setFlag ( )	$\wedge\, (flag = true)$
↓ setX ( )	$\wedge\, (flag = true) \wedge (x = 0)$
↓ decX ( )	$\wedge\, (flag = true) \wedge (x < 100) \wedge \neg(x \geq 1)$

Negation of
underflow-safety condition

Testing using logical formulas

Basic Approach

- Symbolic execution in increasing order.

```
contract Example {
  bool flag;
  uint x;

  constructor () public {
    flag = false;
    x = 100;
  }

  function setFlag () public {
    flag = true;
  }

  function setX () public {
    require (flag == true);
    x = 0;
  }

  function decX () public {
    require (flag == true);
    require (x < 100);
    x = x - 1;
  }
}
```

Goal:
Trigger underflow

Vulnerable Transaction Sequence Found!

SMT Solver:
SAT!

Transaction Sequence	Verification Condition
Constructor ( )	$(flag' = false) \wedge (x' = 100)$
↓ setFlag ( )	$\wedge\, (flag = true)$
↓ setX ( )	$\wedge\, (flag = true) \wedge (x = 0)$
↓ decX ( )	$\wedge\, (flag = true) \wedge (x < 100) \wedge \neg(x \geq 1)$

Negation of
underflow-safety condition

Basic Approach: Symbolic Execution in Increasing Order

1. Enumerate the shortest sequence $t_0 \cdot t_1 \ \cdots \ t_n$.

2. Symbolically validate $t_0 \cdot t_1 \ \cdots \ t_n$.

$\phi =$ **SymbolicExec**$(t_0 \cdot t_1 \ \cdots \ t_n)$ —**SMT Solver**(ϕ)→

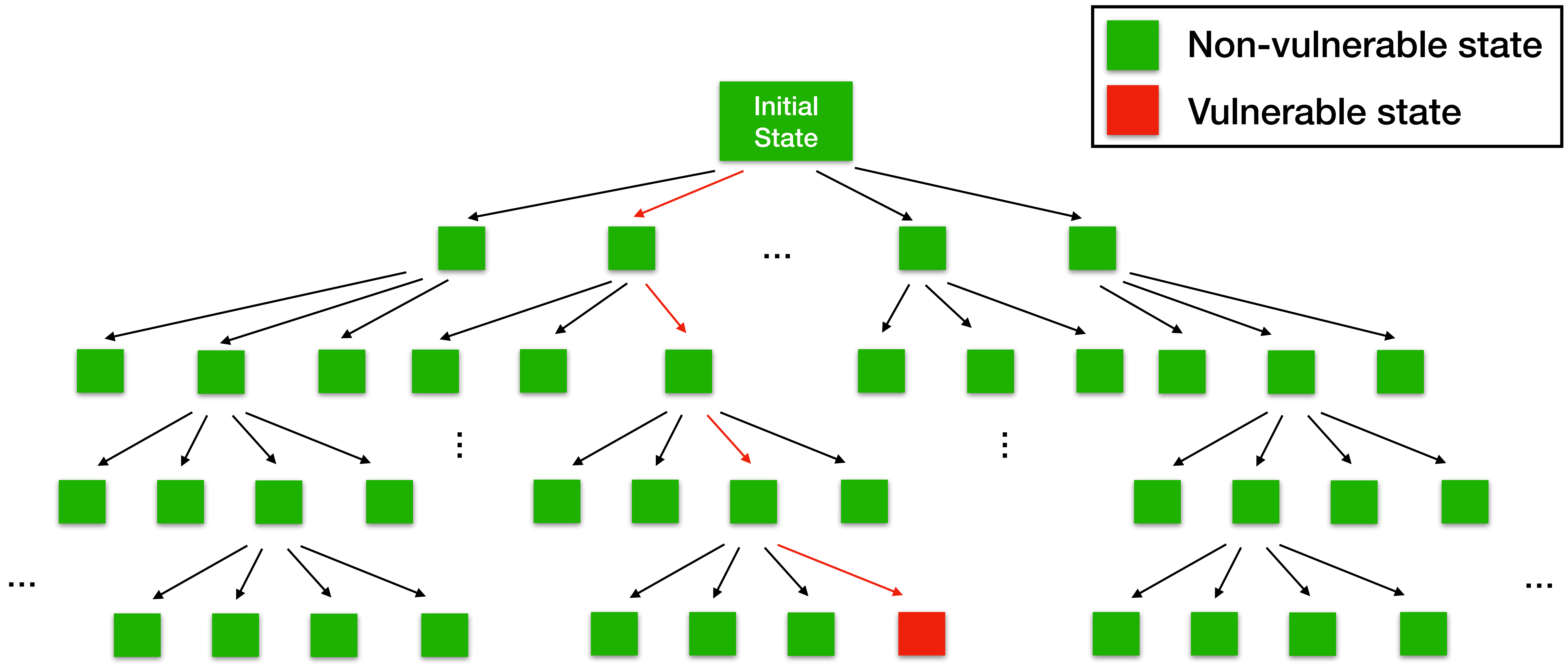
SAT → $t_0 \cdot t_1 \ \cdots \ t_n$ **is a vulnerable sequence!**

UNSAT → $t_0 \cdot t_1 \ \cdots \ t_n$ is NOT a vulnerable sequence!

3. Repeat Step 1, 2 until time limit expires.

Challenge: Path Explosion Problem

- Huge search space for transaction sequences

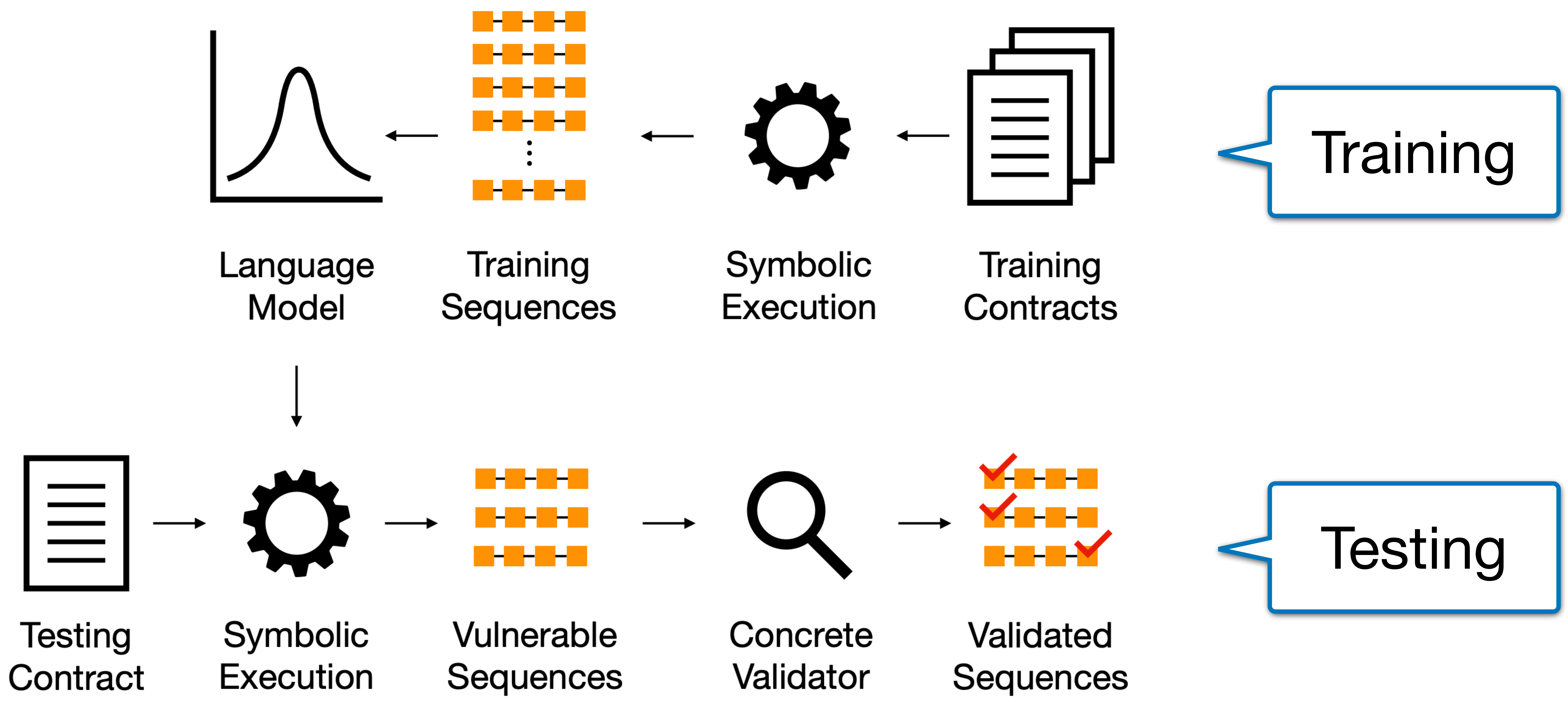


Key Idea:
Language Model-Guided Symbolic Execution

- Prioritize transaction sequences likely to contain vulnerabilities.

`mintToken` → `approve` → `transferFrom` → ... **70%**

`transferFrom` → `approve` → `approve` → ... **0.2%**



Evaluation: SmarTest vs. 5 Tools

Tool	Integer Overflow	Division by Zero	Assertion Violation	ERC20 Standard Violation	CVE Vulnerability Detection (Sample: 242)
SmarTest	1982	203	77	654	219
Mythril (ConsenSys)	460	73	25	n/a	85
Manticore (Trail of Bits)	2	1	3	n/a	0

90.5%

35.1%

Table 1. Results on CVE dataset. Found and validated vulnerable sequences.

Tool	Ether-leaking (90 contracts)			Suicidal (53 contracts)		
	#Contract	#Function	#Line	#Contract	#Function	#Line
SmarTest	**81**	**111**	**111**	**51**	**51**	**51**
ILF [CCS '19]	75	101	n/a	50	50	n/a
Maian [ACSAC '18]	58	n/a	n/a	43	n/a	n/a
teEther [USENIX Security '18]	37	n/a	n/a	n/a	n/a	n/a
Mythril (ConsenSys)	7	8	8	19	19	19
Manticore (Trail of Bits)	9	9	9	3	3	3

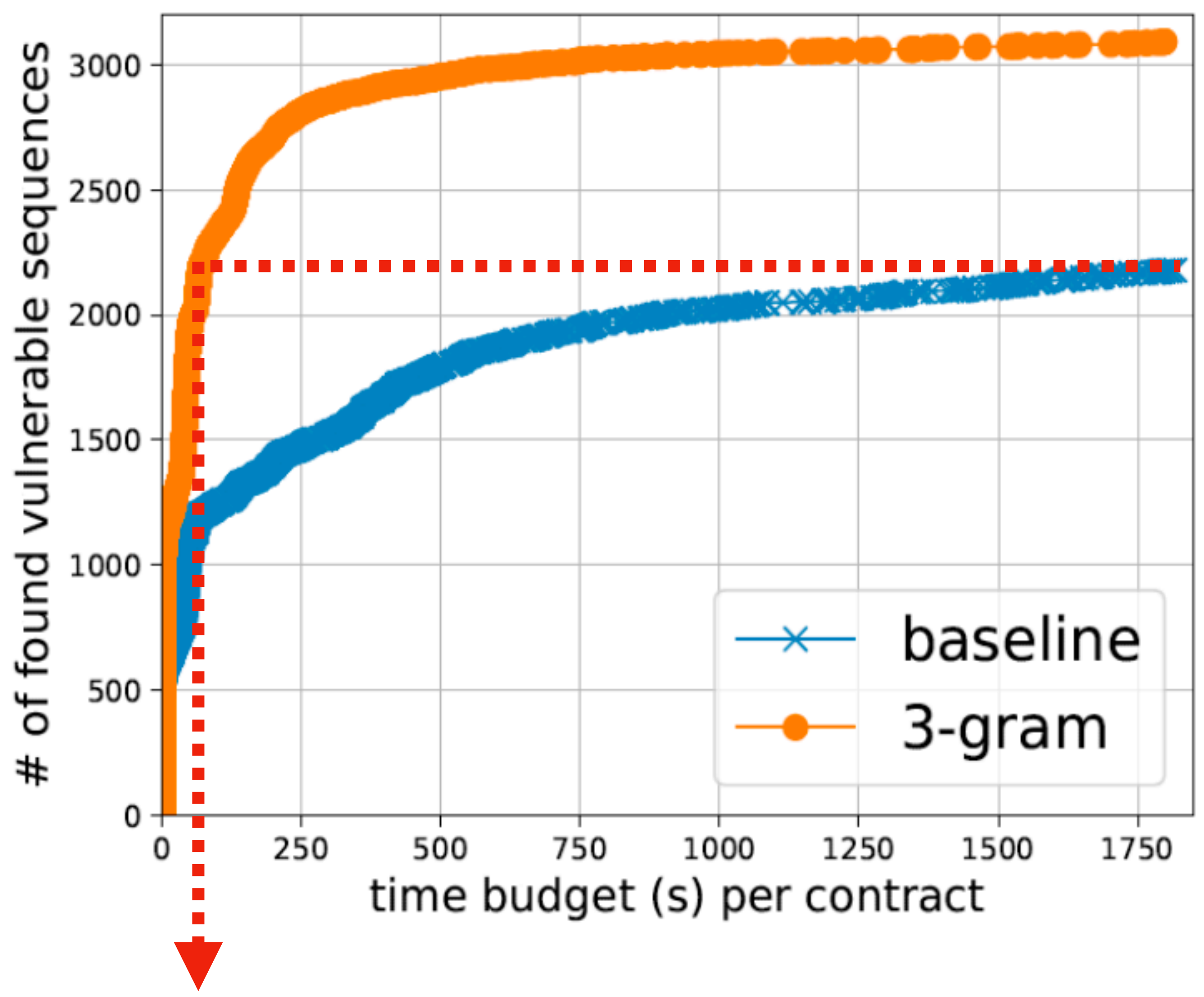
Table 2. Results on Leaking-Suicidal dataset. Found and validated (when available) vulnerable sequences.

Evaluation: Impact of Using Language Model

- **Baseline:** SmarTest without language model
- **3-gram:** SmarTest with 3-gram language model

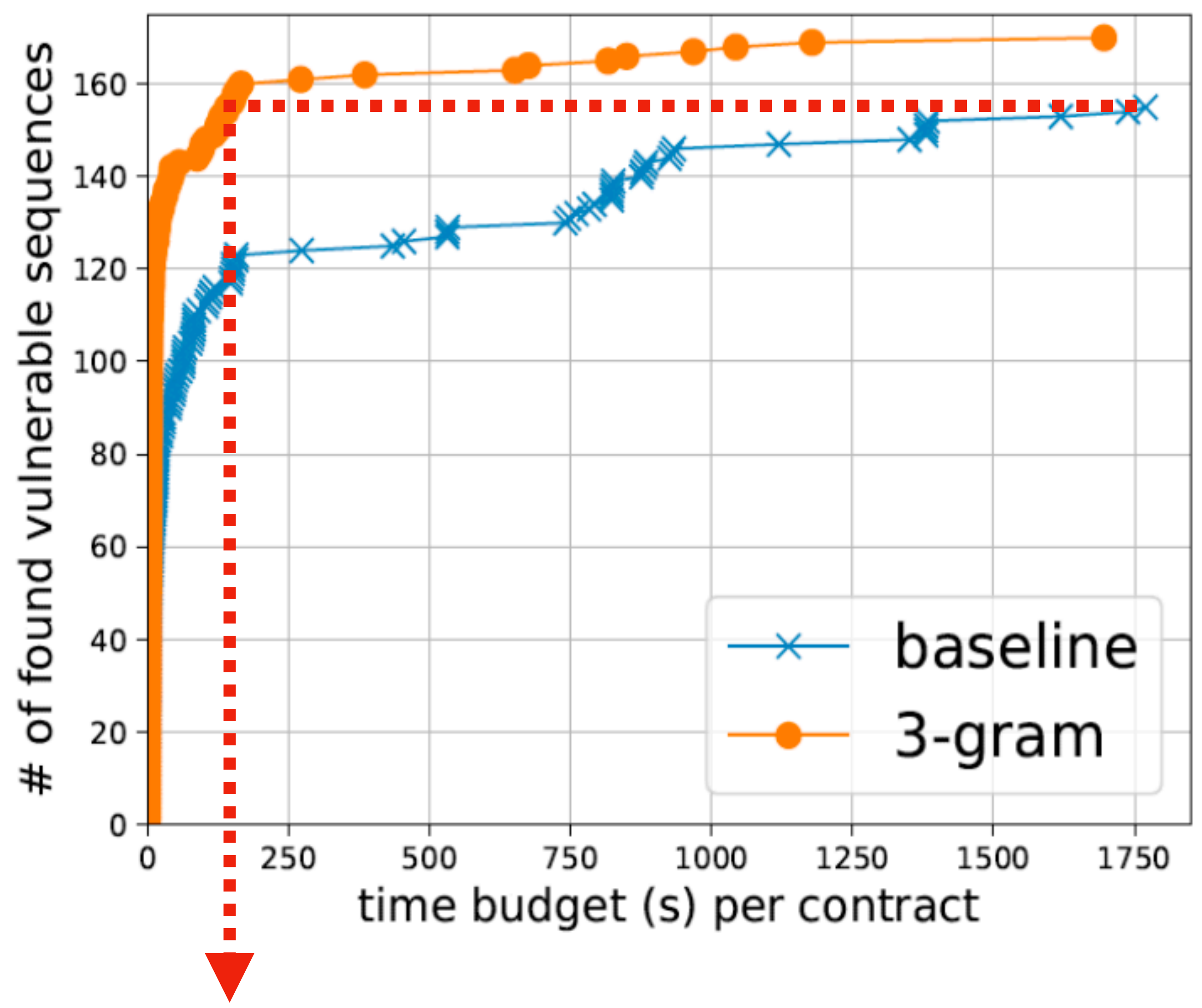
Speed-up
12-30 times

CVE dataset



68s (vs. 1,817s)

Leaking-Suicidal dataset



140s (vs. 1,770s)

Evaluation: Finding Zero-Day Vulnerabilities

- Ran SmarTest on 2,743 contracts from Etherscan.
- Manually confirmed 7 critical vulnerabilities (ERC20 Standard Violation).

Pattern 1: Due to mistakenly named constructors, anyone can tokens for free.

```
1   contract AToken {
2     /* Constructor function */
3     function BToken () public {
4       balance[msg.sender] = 10000000000;
5       totalSupply = 10000000000;
6     }
7   ...
```

"BToken" at line 3 should be changed to "AToken".

Pattern 2: Unrestricted token transfer by unauthorized users.

```
function transferFrom (address from, address to,
  uint value) public returns (bool) {
  require(balance[from] >= value);
  require(balance[to] + value >= balance[to]);
  require(allowed[from][msg.sender] >= value);
  balance[from] -= value;
  balance[to] += value;
  return true;
}
```

Red-colored part is absent in the original code.

3. SmartFix: Program Repair

SmartFix: Fixing Vulnerable Smart Contracts by Accelerating Generate-and-Verify Repair using Statistical Models
FSE 2023

SmartFix's Goal
Fixing Vulnerable Contracts Automatically and Safely

CVE-2018-11411

```
function transferFrom (address from, address to, uint value) returns (bool success) {
  if (value == 0) return false;
  uint fromBalance = balance[from];
  uint allowance = allowed[from][msg.sender];

  bool sufficientFunds = fromBalance <= value;
  bool sufficientAllowance = allowance <= value;
  bool overflowed = balance[to] + value > balance[to];

   if(sufficientFunds && sufficientAllowance && !overflowed) {
    balance[to] += value;                 // overflow
    balance[from] -= value;               // underflow
    allowed[from][msg.sender] -= value; // underflow
    return true;
  }
  else {return false;}
}
```

SmartFix's Goal
Fixing Vulnerable Contracts Automatically and Safely

CVE-2018-11411

```
1   function transferFrom (address from, address to, uint value) returns (bool success) {
2     if (value == 0) return false;
3     uint fromBalance = balance[from];
4     uint allowance = allowed[from][msg.sender];
5
6     bool sufficientFunds = fromBalance <= value;
7     bool sufficientAllowance = allowance <= value;
8     bool overflowed = balance[to] + value > balance[to];
9
10     if(sufficientFunds && sufficientAllowance && !overflowed) {
11      balance[to] += value;                  // overflow
12      balance[from] -= value;                // underflow
13      allowed[from][msg.sender] -= value; // underflow
14      return true;
15    }
16    else {return false;}
17  }
```

Our Goal

Line 6 : Replace <= by >=
Line 7 : Replace <= by >=
Line 8 : Replace > by <

Limitations of Existing Techniques: Simple Patch Only

Rely on a single repair template for each bug type

CVE-2018-11411

```
function transferFrom (address from, address to, uint value) returns (bool success) {
  if (value == 0) return false;
  uint fromBalance = balance[from];
  uint allowance = allowed[from][msg.sender];

  bool sufficientFunds = fromBalance <= value;
  bool sufficientAllowance = allowance <= value;
  bool overflowed = safeAdd(balance[to],value) > balance[to];

   if(sufficientFunds && sufficientAllowance && !overflowed) {
    balance[to] = safeAdd(balance[to],value);
    balance[from] = safeSub(balance[from],value);
    allowed[from][msg.sender] = safeSub(allowed[from][msg.sender],value);
    return true;
  }
  else {return false;}
}
```

sGuard [IEEE S&P'21], SmartShield [SANER'20], Elysium [RAID'22]:
Rely only on inserting runtime checks

※ **safeAdd/safeSub:** raise exceptions if over/underflows occur at runtime

Limitations of Existing Techniques: Simple Patch Only

Rely on a single repair template for each bug type

CVE-2018-11411

```
1   function transferFrom (address from, address to, uint value) returns (bool success) {
2     if (value == 0) return false;
3     uint fromBalance = balance[from];
4     uint allowance = allowed[from][msg.sender];
5
6     bool sufficientFunds = fromBalance <= value;
7     bool sufficientAllowance = allowance <= value;
8     bool overflowed = safeAdd(balance[to],value) > balance[to];
9
10     if(sufficientFunds && sufficientAllowance && !overflowed) {
11      balance[to] = safeAdd(balan [to],value);
12                                  m],value);
13                                  (allowed[from][msg.sender],value);
14      return true;
15    }
16    else {return false;}
17  }
```

To pass line 8:
A: `balance[to] + value >= balance[to]`

sGuard [IEEE S&P'21], SmartShield [SANER'20], Elysium [RAID'22]:
Rely only on inserting runtime checks

※ **safeAdd/safeSub:** raise exceptions if over/underflows occur at runtime

Limitations of Existing Techniques: Simple Patch Only

Rely on a single repair template for each bug type

CVE-2018-11411

```
1   function transferFrom (address from, address to, uint value) returns (bool success) {
2     if (value == 0) return false;
3     uint fromBalance = balance[from];
4     uint allowance = allowed[from][msg.sender];
5
6     bool sufficientFunds = fromBalance <= value;
7     bool sufficientAllowance = allowance <= value;
8     bool overflowed = safeAdd(balance[to],value) > balance[to];
9
10     if(sufficientFunds && sufficientAllowance && !overflowed) {
11      balance[to] = safeAdd(balan [to],value);
12                                     m],value);
13                                     (allowed[from]
14      return true;
15    }
16    else {return false;}
17  }
```

To pass line 8:
A: `balance[to] + value >= balance[to]`

B: `balance[to] + value <= balance[to]`

sGuard [IEEE S&P'21], SmartShield [SANER'20], Elysium [RAID'22]:
Rely only on inserting runtime checks

※ **safeAdd/safeSub:** raise exceptions if over/underflows occur at runtime

Limitations of Existing Techniques: Simple Patch Only

Rely on a single repair template for each bug type

CVE-2018-11411

```
1   function transferFrom (address from, address to, uint value) returns (bool success) {
2     if (value == 0) return false;
3     uint fromBalance = balance[from];
4     uint allowance = allowed[from][msg.sender];
5
6     bool sufficientFunds = fromBalance <= value;
7     bool sufficientAllowance = allowance <= value;
8     bool overflowed = safeAdd(balance[to],value) > balanc
9
10     if(sufficientFunds && sufficientAllowance && !overflowed) {
11       balance[to] = safeAdd(balance[to],value);
12                                    m],value);
13                                    (allowed[from][
14       return true;
15     }
16     else {return false;}
17   }
```

value == 0 in if-branch (by A,B)

To pass line 8:
A: `balance[to] + value >= balance[to]`

B: `balance[to] + value <= balance[to]`

sGuard [IEEE S&P'21], SmartShield [SANER'20], Elysium [RAID'22]:
Rely only on inserting runtime checks

※ **safeAdd/safeSub:** raise exceptions if over/underflows occur at runtime

Limitations of Existing Techniques: Simple Patch Only

Rely on a single repair template for each bug type

CVE-2018-11411

```
1   function transferFrom (address from, address to, uint value) returns (bool success) {
2     if (value == 0) return false;
3     uint fromBalance = balance[from];
4     uint allowance = allowed[from][msg.sender];
5
6     bool sufficientFunds = fromBalance <= value;
7     bool sufficientAllowance = allowance <= value;
8     bool overflowed = safeAdd(balance[to],value) > balance
9
10     if(sufficientFunds && sufficientAllowance && !overflowed) {
11      balance[to] = safeAdd(balan   [to],value);
12                              m],value);
13                              (allowed[from][
14      return true;
15    }
16    else {return false;}
17  }
```

value!=0 after line 2

value == 0 in if-branch (by A,B)

To pass line 8:
A: `balance[to] + value >= balance[to]`

B: `balance[to] + value <= balance[to]`

sGuard [IEEE S&P'21], SmartShield [SANER'20], Elysium [RAID'22]:
Rely only on inserting runtime checks

※ **safeAdd/safeSub:** raise exceptions if over/underflows occur at runtime

Limitations of Existing Techniques: Simple Patch Only

Rely on a single repair template for each bug type

CVE-2018-11411

```
1   function transferFrom (address from, address to, uint value) returns (bo ) {
2     if (value == 0) return false;
3     uint fromBalance = balance[from];
4     uint allowance = allowed[from][msg.sender];
5
6     bool sufficientFunds = fromBalance <= value;
7     bool sufficientAllowance = allowance <= value;
8     bool overflowed = safeAdd(balance[to],value) > balanc
9
10     if(sufficientFunds && sufficientAllowance && !overflowed) {
11      balance[to] = safeAdd(balance[to],value);
12                              m],value);
13                              (allowed[from][
14      return true;
15    }
16    else {return false;}
17  }
```

value!=0 after line 2

Incorrect Patch (Deadcode)

value == 0 in if-branch (by A,B)

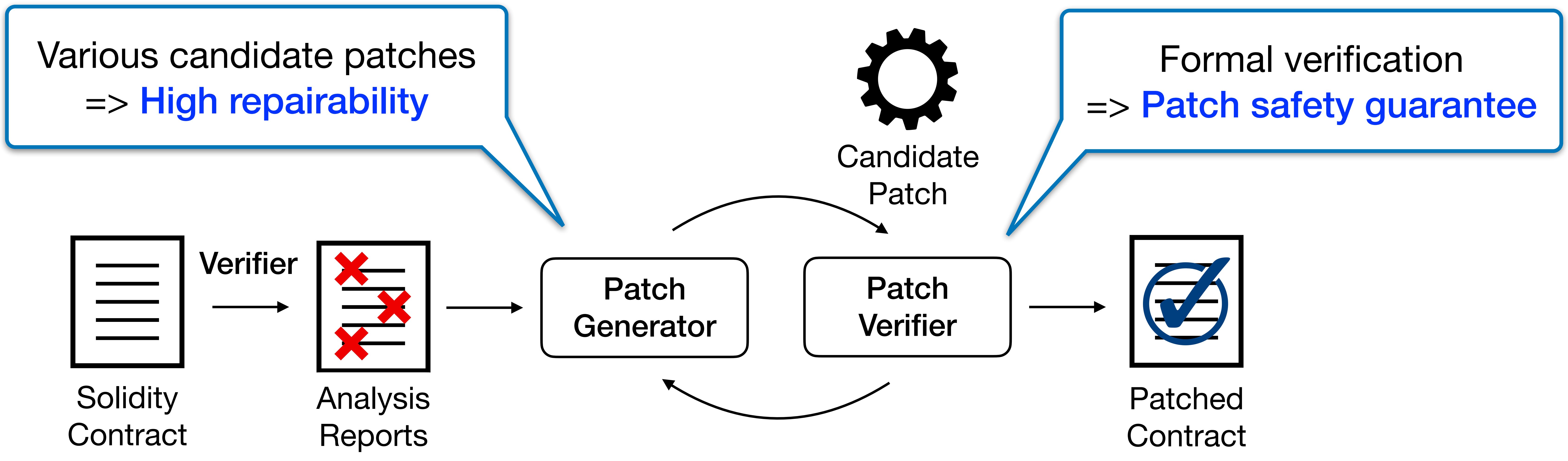
To pass line 8:
A: balance[to] + value >= balance[to]

B: balance[to] + value <= balance[to]

sGuard [IEEE S&P'21], SmartShield [SANER'20], Elysium [RAID'22]:
Rely only on inserting runtime checks

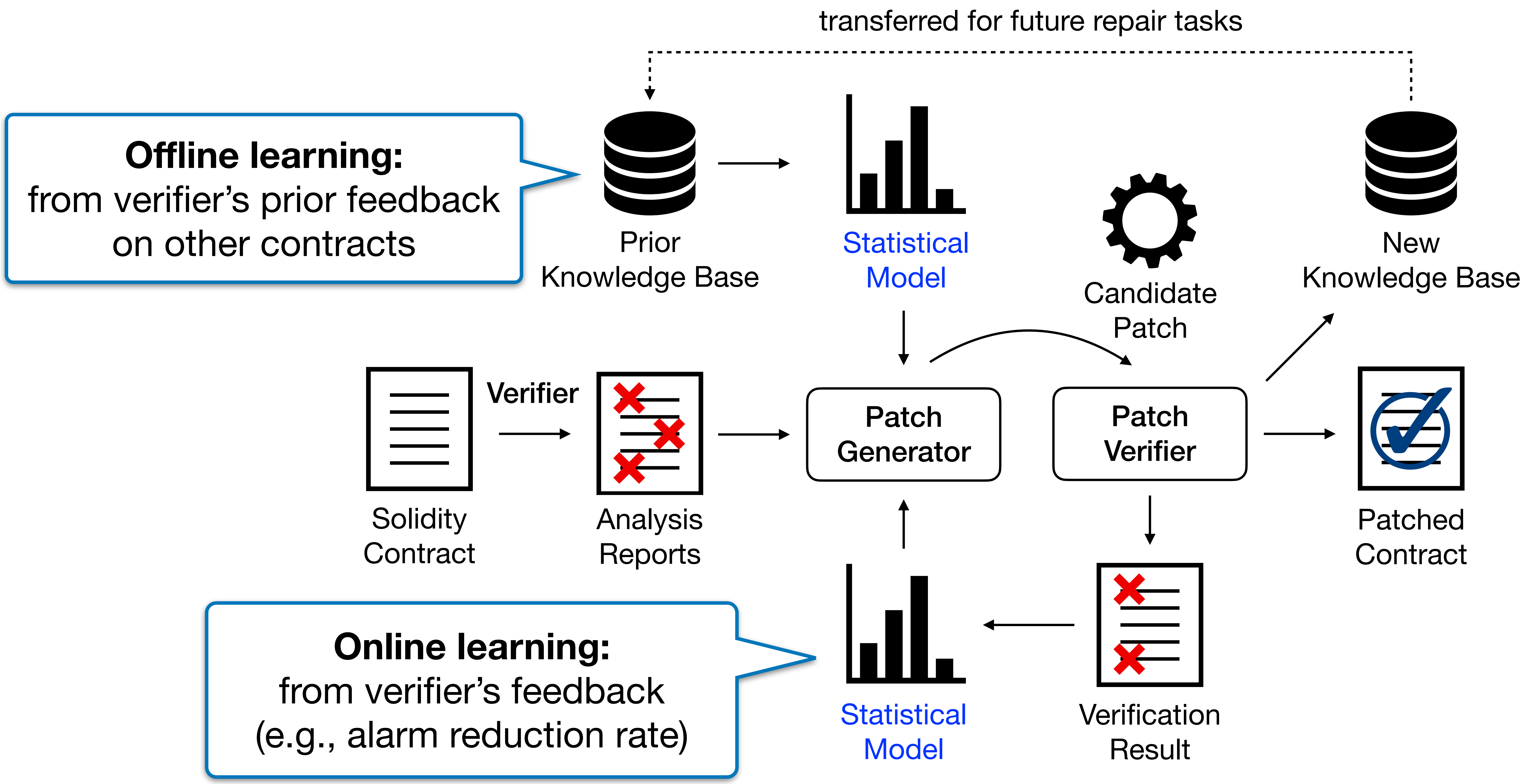
※ **safeAdd/safeSub:** raise exceptions if over/underflows occur at runtime

Basic Approach: Generate-and-Verify



Challenge: Repair efficiency
(large search space + patch verification cost)

SmartFix Approach:
Speeding up Generate-and-Verify using Statistical Models



Key Idea: prioritize likely candidates using learned models

Evaluation: Setup

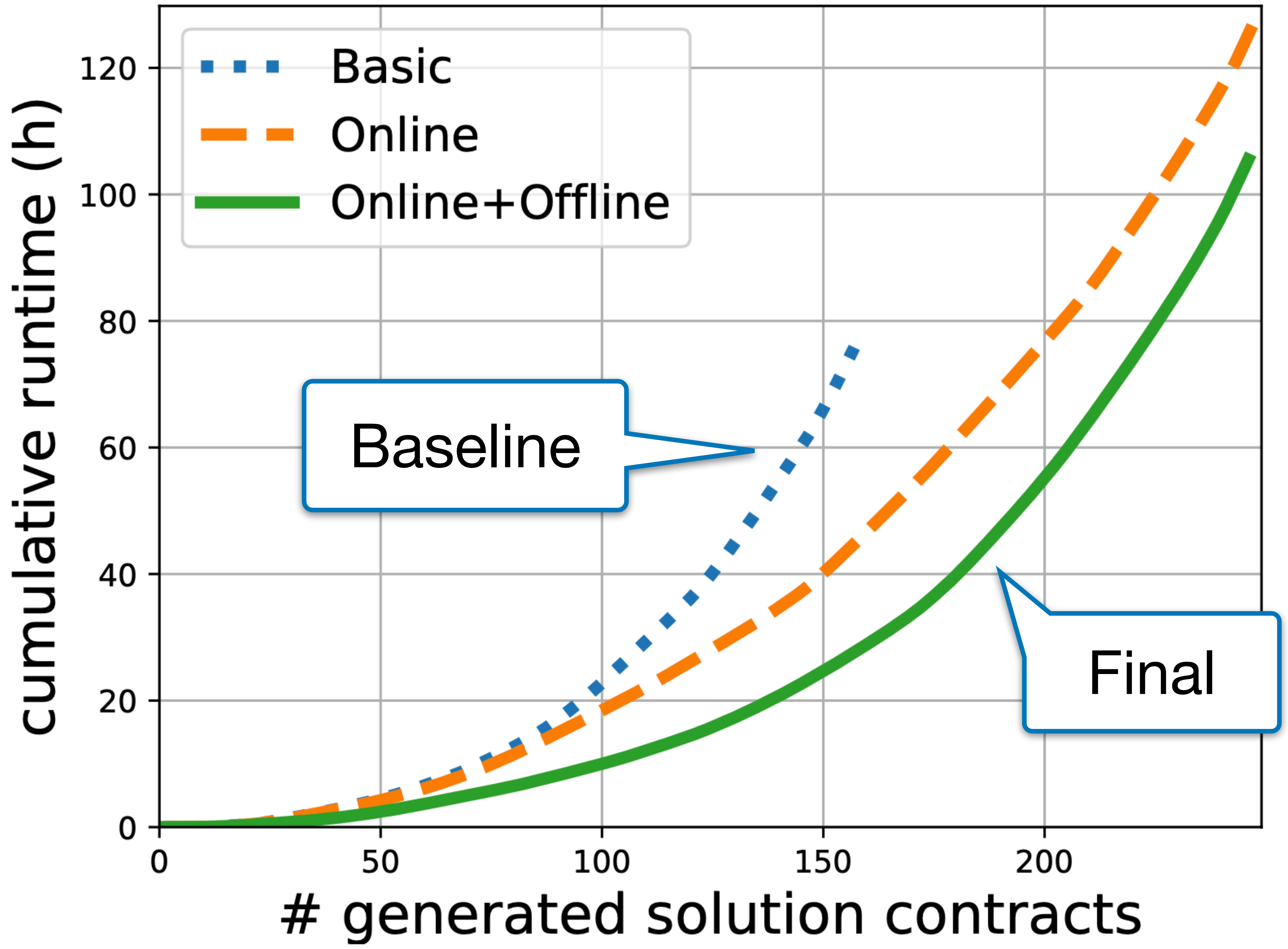
- **Comparison Target:** sGuard [IEEE S&P'21]
  - State-of-the-art fixing tool for Solidity smart contracts.
- **Benchmark:** collected 361 contracts from multiple sources
  - Integer over/underflow: 200 CVE-reported contracts
  - Ether-Leak & Suicidal: 104 from SmartTest [USENIX Sec'21]
  - Reentrancy: 28 from (SODA [NDSS'20],SmartBugs [ICSE'20]) + 17 by bug-injection + 2 from the wild
  - Dangerous tx.origin: 1 from SmartBugs [ICSE'20] + 9 by bug-injection

Evaluation: Fix Rate (vs. sGuard [IEEE S&P'21])

Bug Type	#Bug	SmartFix					sGuard [IEEE S&P '21]				
		#BugRun	#Generated	#Correct	**Fix Succ. Rate**	**Accuracy**	#BugRun	#Generated	#Correct	**Fix Succ. Rate**	**Accuracy**
IO	229	228	218	218	95.6%	100.0%	170	103	103	60.6%	100.0%
RE	52	51	46	46	90.2%	100.0%	33	33	29	87.9%	87.9%
TX	12	12	12	12	100.0%	100.0%	2	2	2	100.0%	100.0%
EL	137	134	83	76	56.7%	91.6%	n/a	n/a	n/a	n/a	n/a
SU	53	51	40	34	66.7%	85.0%	n/a	n/a	n/a	n/a	n/a
IO+RE+TX	293	291	276	276	**94.8%**	**100.0%**	205	138	134	**65.4%**	**97.1%**
Total	483	476	399	386	**81.1%**	**96.7%**	-	-	-	-	-

Fix Success Rate:
94.8% (Ours) vs. 65.4% (sGuard)

Evaluation: Impact of Using Statistical Models



- #Generated Bug-free contracts
  - Baseline (157) vs. Final (246)

Performance Up 56.7%

$$= \frac{246 - 157}{157}$$

Summary:
Program Analysis for Smart Contract Security

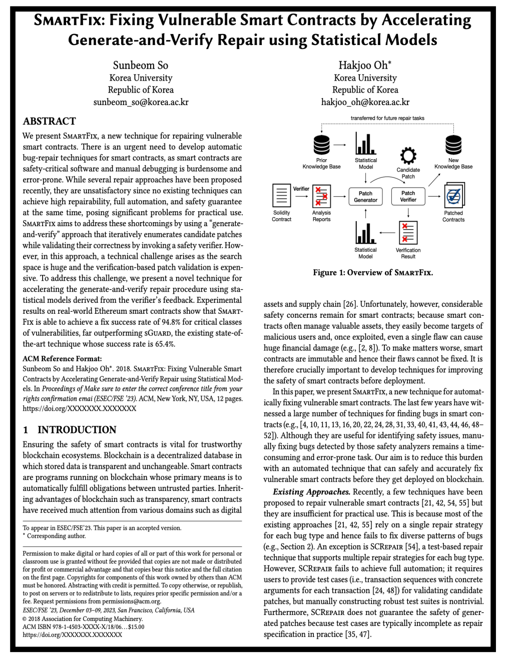
- Existing approaches relied on lightweight, brittle analysis methods.
- Our rigorous, PL-based program analysis (e.g., analysis using logical formulas)
  - **VeriSmart**: precise safety verification using transaction invariants
  - **SmarTest**: symbolic execution for finding vulnerabilities in smart contracts
  - **SmartFix**: generate-and-verify repair for smart contracts

The first successful stories of PL-based approaches for smart contract security

Sourcode & Benchmark: https://github.com/kupl/VeriSmart-public

Research Direction @ GIST

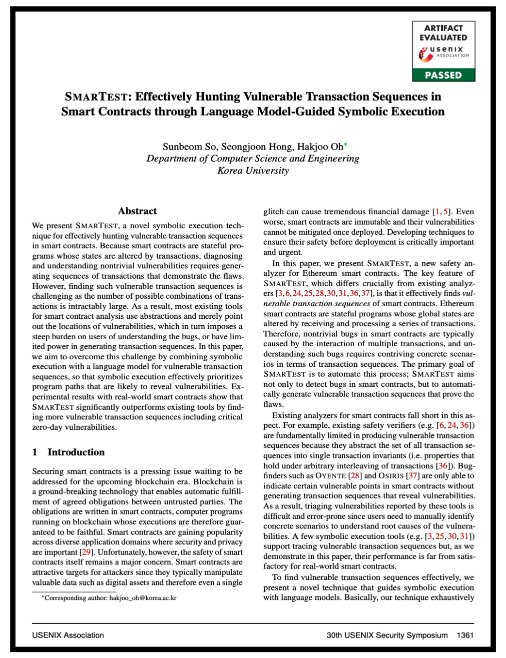
Extensive Experience in Program Analysis



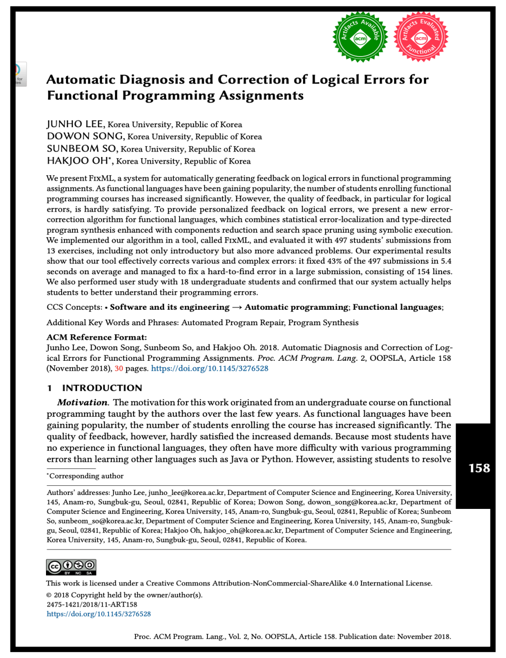
FSE'23

ICSE'23

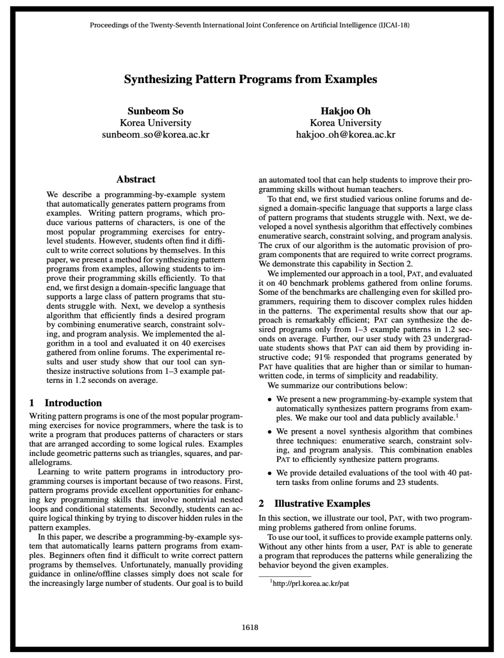
IEEE S&P'20



USENIX Sec'21



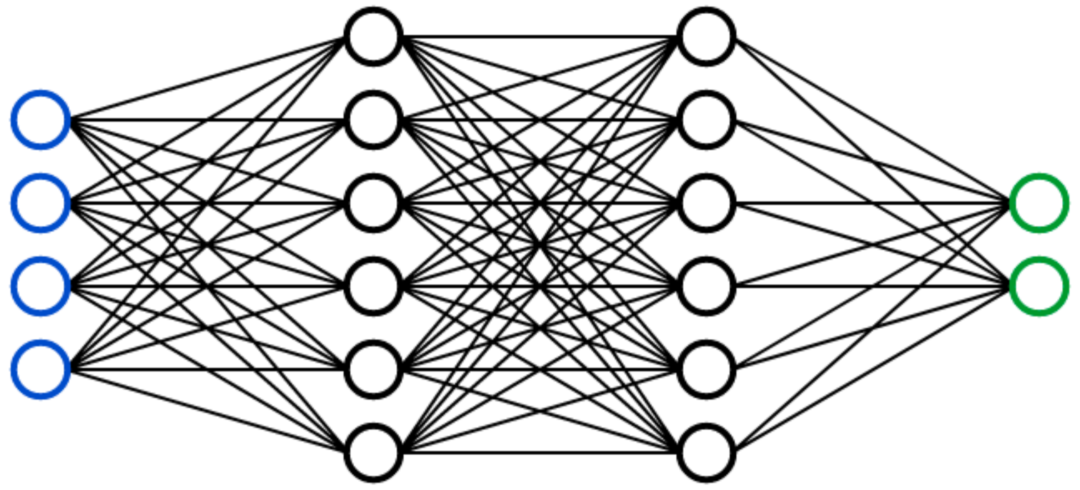
OOPSLA'18



IJCAI'18

…

Safety of
Autonomous Driving System (ADS)



Robustness of
Deep Neural Network (DNN)

…

Thank you!